Exhibit 12

An Approach to Reducing Delays in Recognizing Distributed Event Occurrences

Madalene Spezialetti
CSEE Department, Packard Lab
Lehigh University
Bethlehem, PA 18015

Abstract.
In order to aid in analyzing the activity of distributed computations, monitoring systems have been developed which allow a user to specify behavior of interest in the form of an event definition. The monitoring system then analyzes the behavior of the computation, notifying the user when the computation exhibits the behavior described by a definition. To accomplish such an analysis, the activity of the components of an event must be monitored individually and information pertaining to their values sent to a designated monitor for evaluation. Due to the need to accumulate state information, delays often result between the occurrence of activity specified by an event definition and the detection of the activity by the monitoring system. This paper presents an approach to reducing delays in event recognition by incorporating knowledge about the characteristics of the behavior to be recognized into the evaluation protocol. These characteristics are used in the development of techniques to localize information required for event recognition at the place of evaluation, thus decreasing the impact of communication delays on the recognition task and leading to a reduction in the time which elapses between an event's occurrence and its recognition.

1. INTRODUCTION

As distributed and parallel computing capabilities increase, there is a growing need to develop techniques which aid in analyzing the behavior of such systems. One approach to aid in this analysis allows users to describe activity of interest in the form of *event definitions* [1,2,5,8,10,11,13,15]. These definitions are predicates which test the states of various system elements. The activity of the elements specified by an event definition is monitored, and data pertaining to their activity collected for analysis. This data is organized into a view of the computation state, and the event definition is evaluated in terms of the states of its operands. An event *occurs* at the point that the behavior of the computation fulfills the specification of its definition, that is, at the point that the application of the definition's operators to the states of the operands would yield a TRUE result. An occurrence of the event is *recognized* at the point that the monitoring system detects its occurrence.

Due to the decentralized nature of distributed computations, the task of event recognition often requires the transmission of state information from a processor where a monitored element resides to another processor for evaluation. This communication can lead to an unpredictable delay between the occurrence of the activity specified in an event and the recognition of its occurrence. In many situations, such as process control and various real time applications, it is desirable to reduce the delay between event occurrence and recognition. In addition, the use of a centralized location for the processing and evaluation of event information can lead to message congestion, potentially delaying the event recognition further as well as possibly impeding the activity of the monitored processes.

A time lapse will always result between an event's occurrence and its recognition due to the need to evaluate the event. This delay can be minimized if the only delay between an event's occurrence and its recognition is that which results from the actual evaluation. Such a recognition will be referred to an *Immediate* recognition. If there is a delay, due to the need to accumulate additional data from other processors, between the point that the event occurs and its occurrence is recognized, the recognition will be referred to as a *Delayed* recognition.

This paper addresses the task of reducing the delay between an event's occurrence and its recognition by applying techniques which attempt to recognize an

event Immediately. Techniques are presented which address the problem of monitor placement. The characteristics of an event are utilized to determine if an allocation of the monitoring task can be made which would guarantee the Immediate recognition of the event's occurrence. If such an allocation can not be made, knowledge pertaining to the stability of the elements tested by an event are incorporated into analysis techniques in order to allow the re-use of information in event evaluations. These techniques attempt to localize information at the place of evaluation in order to reduce the impact of communication delays on the recognition of event occurrences.

The next section provides a description of events and of the various monitoring modules which cooperate to recognize events occurrences. Section 3 describes the event characteristics which will be utilized to aid in the reduction of recognition delays. Section 4 briefly discusses the characteristics of Immediate recognitions. The task of determining if an Immediate recognition can be guaranteed through the placement of monitors based on the examination of an event is addressed in Section 5. Section 6 presents a means of reducing the recognition delay by re-using information in successive evaluations based on the characteristics of the monitored elements. The integration of these techniques into a protocol for forming views of a distributed computation's state and examples demonstrating the application of the techniques are presented in Sections 7 and 8, followed by the conclusion in Section 9.

2. EVENTS AND MONITORS

An *event definition* is a description of the activity which is to be detected by a monitoring system. Specific definition languages which provide various forms of abstraction have been developed [1,2,5,10,11,13]. However, since this paper concentrates on general recognition problems, the existence of a general purpose specification capability, similar to that used in [1], is assumed. In particular, the existence of a set of *primitive events* is assumed, where primitive events are the (application specific) lowest level of system activity which may be used in the construction of event definitions. Intuitively, primitive events would be available to test such entities as communication transmission or reception, memory and file accesses, process or entity state and process status.

An event definition, is assumed to be a predicate whose operands, or *components*, are combined or tested using relational, logical or temporal operators. Events may be hierarchical in nature, so that components may be primitive events or events which were defined by the user (thus, references to components can also be applied to events and *vice versa*). It is also assumed that a set of *event operators*, which take events as their operands, are available to test the value of the events which are their operands. The event operators are comprised of the set of logical operators {AND, OR} which test the truth value of the operands, and the set of temporal operators {|,#,',} which test the occurrence or order of occurrence of the operands. The semantics of the event operators are given in Figure 1.

LOGICAL OPERATORS

E1 **AND** E2	TRUE while both E1 and E2 are simultaneously TRUE.
E1 **OR** E2	TRUE while either E1 or E2 are TRUE.

TEMPORAL OPERATORS

E1 # E2	TRUE when both E1 and E2 have become TRUE.
E1 | E2	TRUE when either event E1 or event E2 becomes TRUE.
E1 [E2	TRUE when E2 becomes TRUE after E1 has become TRUE.

Figure 1. Event Operators.

In order to recognize an event occurrence, data regarding the states of the components must be collected and evaluated. Given the absence of a global clock or a single shared memory in a distributed system, some method must be provided by which the information can be organized into consistent representations of the system state [9]. The responsibility for maintaining a protocol to enable such a view of a distributed computation must also be integrated with the event recognition system. While the notion of a global clock or shared memory does not exist for a distributed system at large, such a notion is available on an individual processor within the system. The organization of monitoring modules and delegation of monitoring responsibilities utilizes this notion in order to reduce the overhead of defining consistent views of the system and to reduce the time required to collect the information pertaining to the current states of components located at a specific processor.

At each processor at which a component of a particular event is located is an *event monitor*. This monitor will maintain information pertaining to the states of each component of the event which is located at that processor. Associated with each process is a *component*

monitor, which is responsible for detecting changes to the monitored components within that process and, when an alteration occurs to a component, transmitting the value of that alteration to the appropriate event monitor. It is assumed that, when a change occurs to a monitored component, the operation of informing the event monitor of that change is atomic. In that way, the event monitor located at a particular processor maintains information regarding the current values of all of the event's components at that processor. Event monitors are also responsible for maintaining whatever inter-processor protocol is required in order to obtain a consistent view of the component states at various processors.

Evaluation monitors must also be designated for an event. An evaluation monitor may be assigned the task of evaluating an entire event or some portion of an event and is responsible for accumulating and organizing the data which is required to perform the evaluation and potentially make recognitions. An evaluation monitor may be placed at any processor, even one at which none of the components or elements associated with the monitor's event reside. However, such an arrangement would always necessitate some data transmission in order to perform an evaluation and therefore would always result in some delay, due to interprocessor communication, between an event's occurrence and its recognition. To reduce such a delay, evaluation monitors are more likely to be located at a processor at which some component of the event resides. In such a case, the evaluation responsibilities are incorporated into the event monitor at that processor. In that way, information pertaining to the components located at that processor is readily available for use in any evaluation. However, for generality, whenever discussing the assignment of evaluation responsibilities, the module will be referred to as an evaluation monitor, although it may serve as an event monitor as well.

3. EVENT CHARACTERISTICS

Given that the components of an event to be recognized will be scattered throughout a system, the information regarding the changes to their states must be communicated to the evaluation monitor or monitors responsible for recognizing the event occurrence. It is the communication of component state information to evaluation monitors which must be analyzed in order to determine if a recognition can be characterized as Immediate and it is this communication, and its impact on the monitoring task, which the techniques described in this paper attempt to reduce. The characteristics of the event to be recognized and of its individual components are utilized to achieve this reduction. Two characteristics are primarily utilized by these techniques: *the identification of critical components* and *component stability*.

3.1. CRITICAL COMPONENTS

Many factors contribute to the determination of when an event occurrence can be recognized. Such factors include communication delays due to the need to accumulate monitoring information and the time required for event analysis and verification. However, while various considerations influence the recognition of an event, the ability to isolate the point at which the event becomes TRUE is a major factor in determining when an event occurrence can be recognized.

A change to the value of a single component, change(c_i), or simultaneous changes to the value of two or more components of an event, change(c_i, c_j...), constitute a change in the value of the event itself. The *monitoring* of a computation in order to detect an occurrence of an event can be viewed as a series of (change(e) :: eval(e)) pairs, where each change(e) is associated with either a change in the value of one component of e or simultaneous changes to two or more components of e:

$$(change_1(\mathrm{e})::eval_1(\mathrm{e})),\ (change_2(\mathrm{e})::eval_2(\mathrm{e}))...$$

where for $change_i$, $change_j$, i<j implies $change_i$ preceded $change_j$, under the consistent view of the system activity utilized by the monitoring system. A component c_k is a *critical component* of an event, *e*, if

$$change_i(c_k) :: eval_i(\mathrm{e}) = \mathrm{TRUE}$$
$$\mathrm{AND}$$
$$change_{i-1}(c_j) :: eval_{i-1}(\mathrm{e}) = \mathrm{FALSE}$$

that is, the critical component is the component whose change causes an occurrence of the event. Since the occurrence of the critical component signals the occurrence of the entire event, it follows that the recognition of the critical component's occurrence influences the recognition of the event. In particular, *the time lapse between the occurrence of an event and its recognition can be no less than the time lapse between the occurrence of the critical event and its recognition.* However, it should be noted that while it is beneficial to be able to identify the critical component of an event, it is not always possible.

3.2. STABILITY OF EVENT COMPONENTS

The second event characteristic which will be utilized to aid in the reduction of recognition time is stability, or *monotonicity*. Given that event components are dispersed throughout a system, the value of a component may change between the time that its value is sent to an evaluation monitor and the time that the value is used in

an evaluation. By recognizing the stable properties which may be exhibited by the components of an event, it is possible to infer the current state of a remote component's value. This knowledge can allow information regarding a component's state to be re-used, thus reducing the need to wait for current state data for each evaluation. Thus, the ability to guarantee the stability of component values affects the ability to reduce the time required to recognize an event occurrence. A component can be classified as **monotonic, non-monotonic** or **dependent monotonic.**

3.2.1. MONOTONICITY

A component is **monotonic** if, upon reaching a particular value, it retains that value from that time onwards. Let $eval_t(c)$ represent the value of a component c at time t. Component c is monotonic if there exists some *holding value, hv*, such that if there exists some i, for which $eval_i(c) = hv$, then for all j, $j>i$, $eval_j(c) \equiv eval_i(c) \equiv hv$. In general, monotonic components test if a certain situation has ever *occurred.* For example, the event *(Process A accessed file F)* is monotonic since it will be TRUE at the point that file F was accessed by process P and will remain TRUE thereafter. An event is *true monotonic* if its holding value is TRUE. All references to monotonic events henceforth in this work are assumed to refer to true monotonic events.

A component which exhibits monotonic properties can be either **intrinsically monotonic** or **operator monotonic.** The intrinsic monotonicity of a component is determined at the primitive event level, that is, a primitive event which fulfills the monotonic criteria is intrinsically monotonic. In general, such primitive events test the occurrence of an action. Examples of intrinsically monotonic primitive events are *(process A reached line 1000), (process B wrote to file F)* and *(process C has terminated).* Since each of these primitive events test if an action has taken place, the event will become TRUE when the activity occurs, and will not revert back to FALSE.

For operator monotonic components, the operator for which the component is an operand determines its monotonicity. In particular, although a component may not be intrinsically monotonic, it may be associated with an operator which indicates an occurrence test. Although the component may be testing the current value of an object (an intrinsically non-monotonic characteristic), the operator causes the operand, and thus the expression, to become monotonic. Thus operands which, taken in isolation, are not monotonic can fulfill the monotonic criteria due to the operator with which they are associated. In particular, temporal operators, which test the order of occurrence of events, impart monotonicity on their operands. That is, once the component has become TRUE, its occurrence would have fulfilled its contribution to the event. Even if the value of the component itself may later change, its contribution toward the event's recognition will remain stable. Further, since the operator imparts monotonicity on its operands, an expression formed with a temporal operator is monotonic as well, that is, once the specifications of its components, as they contribute to a particular recognition, have been fulfilled, the expression will become and remain TRUE.

Given the stability of monotonic components, the knowledge that such a component has reached its holding value can be utilized to reduce the amount of data which must be processed in order to recognize an event occurrence. In particular, once the holding value has been communicated to a monitor responsible for recognizing an event of which it is a part, that value can be used in repeated evaluations of the event until an occurrence of the event has been recognized. Since values which the evaluation monitor already possesses can be utilized in evaluations, the use of monotonic information can lessen the delay between the occurrence of an event and its recognition. In the best case, all but one of the monitored components will remain in the protocol, reducing the task of recognizing a distributed event to a local recognition of its critical component's occurrence, thus eliminating any delay due to interprocessor communication between the event occurrence and its recognition.

3.2.2. NON-MONOTONICITY

Components which do not exhibit the stable properties of monotonicity are termed **non-monotonic.** A non-monotonic component is a component whose value can change at any time. No assertions can be made with respect to the length of time for which a non-monotonic component will maintain its current value nor can any projections be made as to its future value. For example, the event ($A.x$<3), which tests the value of a variable x at a process A, is non-monotonic. As the value of $A.x$ changes, the evaluation of ($A.x < 3$) will also change to reflect the values assumed by $A.x$. Thus, based on the current value of the evaluation, no predictions can be made as to the future values which ($A.x$<3) may assume. Events which contain non-monotonic components are more difficult to recognize, and their recognition tends to be far more communication intensive. Since the value of the events may change at any time, a greater need arises to inform other monitors of the fluctuations in the their values. Further, given the instability of a non-monotonic component, it is not possible, in general, for a monitor to be assured that the current value of a remote component corresponds to the

information which the monitor possesses.

3.2.3. DEPENDENT MONOTONICITY

While non-monotonic components are volatile in nature, there is a bridge between monotonic and non-monotonic components, namely, **dependent monotonic** components. Dependent monotonic components are components which are non-monotonic by nature but which exhibit monotonic characteristics over certain known periods of time, referred to as *holding spans.* During the holding span, the component will maintain a holding value. Dependent monotonicity enables the efficiencies which can be gained though monotonicity to be utilized in recognizing events whose components are inherently non-monotonic. For any dependent monotonic component, a **dependent element** can be identified. The dependent element is that entity which determines the delineation of the holding span, that is, that element which controls the period during which the component will maintain its holding value. A dependent element could, for example, be a synchronization element, such as a semaphore, an activity which involves inherent synchronization, such as synchronous communication, or a module or element to which control is explicitly transferred, such as module which is invoked via a remote procedure call (RPC) facility [3].

Unlike monotonic or non-monotonic components, which can be classified by inspection, the determination of an event's status as dependent monotonic can be determined either by the nature of the component itself, by the activity of the process associated with the component or by the environment in which the component's event can occur. A component is **explicitly dependent monotonic** if its dependent monotonicity can be identified via the examination of the event. Such components imply an explicit transfer of control or suspension of processing by their nature. An example in explicitly dependent monotonicity is the event *(A suspended on semaphore S).* The event itself implies a suspension of activity of process A and identifies the dependent element, *S*, which will control the period over which this event would remain in the holding value of TRUE.

A component is **implicitly dependent monotonic** if it is non-monotonic in nature, but it assumes monotonic characteristics due to the behavior of a computation. The dependent monotonicity of such a component can not be determined by the inspection of the definition. Rather such factors as the transfer of control, or access to a synchronization or locking facility by the process at which such a component resides, indirectly causes the component to enter a holding span. Thus, it is necessary to examine the control flow and synchronization mechanisms utilized by the component processes in order to determine if a component is implicitly dependent monotonic and the holding spans over which the component behaves in that manner. As an example, consider the event *((A.x < 3) AND (B.y > 5))*, in which both of the components are non-monotonic. If during the course of the computation, process A makes a remote procedure call to process B, the component *(A.a < 3)* can be guaranteed not to change, that is, it will behave monotonically, from the time that the call is made until control is returned to A.

4. REDUCTIONS IN RECOGNITION TIME

The recognition of an event occurrence involves the collection of required component information, the organization of this information into consistent views of the computation's activity, and the evaluation of the event definition based on the component values. The approaches to the reduction of recognition time presented in this paper attempt to minimize the impact of communication delays on the evaluation procedure. If the impact of communication delays on the recognition of an event can be totally eliminated, then the recognition is termed *Immediate.* That is, the of an event, *e*, is characterized as Immediate it there is no delay, due to monitoring related communication, between the occurrence of an event and its recognition. Thus, a recognition is Immediate if it meets the following criteria:

1) The evaluation monitor responsible for recognizing the occurrence of *e* is located at the same processor as the critical component of *e*.
2) All state information required to recognize the occurrence of *e* is available at the evaluation monitor at the point that the change occurs to the critical component which, in turn, constitutes an occurrence of the event *e*.

The techniques presented in this paper to reduce recognition delays attempt to accomplish Immediate recognitions of events. Although such a recognition can not always be achieved, the techniques can still lead to a reduction in the time lapse between an event's occurrence and its recognition. These approaches focus on two aspects of the recognition task: the placement of the evaluation monitor relative to critical components and accumulation and the utilization of component information.

5. EVALUATION MONITOR PLACEMENT

In order to recognize an event occurrence Immediately, evaluation monitors which are capable of recognizing an event occurrence must be located at the critical components of the event. Thus, the task of monitor placement attempts to assign evaluation responsibilities so that they may be located at the critical components,

leading to a decentralization of the evaluation task. The first step in determining if a placement of evaluation monitors is possible which would permit an Immediate recognition involves the analysis of event operators. In particular, for each operator, a determination is made as to whether or not, due to the semantics of the operator, it is possible to identify a critical component for that expression. In this analysis, a definition will be viewed as an expression tree. Initially, each leaf node represents a component, and each interior node represents an operator, and thus a potential point of evaluation. A link between a parent node, p, and a child, c, indicates that data must be transmitted from the processor where c is located to the processor where p is located in order for p to perform an evaluation. The placement of evaluation monitors is represented by grouping an operation with one or more of its children, indicating that an evaluation monitor responsible for performing the specified operation will be placed at the same processor as the child with which it is grouped. The tree is modified using the following rules to group interior nodes p, and their children c_1 and c_2:

PR1 If, due to the semantics of the operation represented by p, both c_1 and c_2 can be designated critical components, p is *split*, such that the operator represented by p is grouped with both c_1 and c_2, and p', the parent of p, becomes the parent of both c_1 and c_2. The splitting of an evaluation node indicates that an instance of the evaluation monitor associated with p is placed at the processors where c_1 and c_2 are located, thus insuring that the evaluation monitor is placed at the critical components.

PR2 If, due to the semantics of the operation associated with p, one child, c_i, can be designated a critical component for the expression, then the child c_i, is *absorbed* into p, that is c_i is associated with the parent node, p, and the node formerly associated with c_i is pruned.

PR3 If due to the semantics of the operation associated with p, neither c_1 nor c_2 can be designated a critical component for the expression, then one of the children is absorbed into the parent node p. In this situation other criteria, such as the location of other components of the event or the desire to centralize or decentralize the evaluation, may be utilized to determine which child of p should be absorbed. While the choice of which child to absorb may effect the amount of monitoring messages actually needed to recognize the event, since the critical component can not be determined via examination of the event, the choice of either component does not effect the decision as to whether or not an Immediate recognition can be guaranteed.

The above rules are iteratively applied until there are no remaining operators which have not been associated with one of the processors at which a basic component is located. An Immediate recognition of the event e can be guaranteed by applying the placement of evaluation monitors as indicated by the transformed expression tree if either:

1) All links have been removed from the transformed expression tree or
2) For every group of nodes connected by links, all nodes of the group are located at the same processor.

If the above criteria are met, then it can be guaranteed that, for each potential critical component of an event, all information required to recognize an occurrence of the event would be present at the evaluation monitor located at the same processor as the critical component at the time of the critical components occurrence. Thus, at any time that a change should take place to a critical component such that its new value would constitute an occurrence of the event, it would be possible to recognize the event occurrence based solely on information available to the evaluation monitor located at the critical component at that time.

Let us consider the application of the above algorithm, based on the operators described above, to the expression c_1 *op* c_2. An expression formed using one of the operators {|,OR, or}, will become TRUE when *either* of its operands becomes TRUE. Since both components individually constitute known critical components, PR1 is applied to these operators. The operator {'}, implies that c_2 occurs after c_1 and therefore will always be the critical component. Thus, PR2 would be applied to the operator {'}. The states of both components determine the value of an expression formed with any of the operators {AND,#,and,<,>,<=,>=,=,<>}. Thus, PR3 would be applied to these operators.

Consider the event *(((B in critical_sec) OR (C in critical_sec)) & (A blocked on receive)) | (D in critical_sec))*, where processes A and B are located at processor P_1, process C is located at processor P_2 and process D is located at processor P_3. It is assumed that the critical section tests are made by local program counter checks, thus the values of all components are determined based on the status of the individual processes. The events are represented in the figure using the process names only. The initial expression tree is shown in Figure 2a. The OR operator is split between its operands, resulting in the configuration in 2b. The AND operator however can not be split, and therefore is

grouped with one of its children, *(A blocked on receive)*, as shown in 2c. Finally, operator I is split between its children, resulting in the final tree in 2d. Since a link exists between *(B in critical_sec)* and *(A blocked on receive)*, and these components are not located at the same processor, an Immediate recognition of this event can not be guaranteed. However the application of the algorithm indicates a placement of monitors which will localize the evaluation of components where possible, to decrease the amount of state information transmission and thus increase the likelihood of an Immediate recognition when possible.

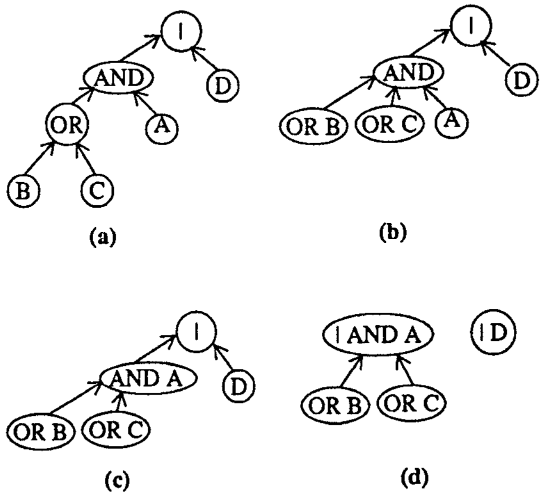


Figure 2: Grouping of Components and Operators

It should be noted that even if a transformed expression tree does not meet the above criteria, an Immediate recognition of the event's occurrence is not necessarily precluded. Due to the order in which the monitored activity occurs and the availability of component information, Immediate recognitions of the event may occur. However, the inability to meet the criteria does imply that such a recognition can not be guaranteed in advance based solely on the semantics of the event. The next section presents techniques which attempt to achieve Immediate recognitions on-the-fly as a means of reducing the time lapse between an event's occurrence and its recognition in situations where an Immediate recognition can not be guaranteed in advance. These techniques utilize the knowledge of components' monotonicity to permit the re-use of information which an evaluation monitor may possess.

6. ON-THE-FLY IMMEDIATE RECOGNITIONS

In the absence of a global clock, it is not generally possible to construct a view of the system state as it existed at a specific point in time. Thus, in order to evaluate the state of a computation for the purposes of event recognition, some method must be employed by which the component state information can be formed into a consistent representation of the system state. Techniques have been developed which provide such a view of system activity [4,6,7,9,12,14,16]. Thus, for current purposes, it will be assumed that the monitoring system is provided with a series of consistent *views*, v_1, $v_2 ... v_n$, of the monitored components' states. These views are constructed at the evaluation monitor based on the reception of a value for each component of an event which is tagged with a view number, indicating the view of which the component value is a part. All component values of a particular view are considered to be "concurrent".

To reduce the time lapse between the occurrence of an event and its recognition, the evaluation monitor maintains information regarding the monotonicity of components whose state information it possesses so as to allow the re-use of the information in subsequent evaluations. *Current_view_number*, indicates the view in which the current evaluation of component information is taking place and the view in which an event occurrence will be recognized if the definition evaluates to TRUE. The use of information in the evaluation of an event e for a particular view is determined based on the following rules:

EV1 If the value of a component c_x is received for a view v_j, such that c_x is either inherently monotonic or operator monotonic and its value, designated $c_{x,j}$, is the holding value of the component, the value $c_{x,j}$ can be used in the evaluations of any subsequent views v_k, k >= j.

EV2 If the a component c_x is dependent monotonic and known by the evaluation monitor to be within a holding span, the last value assumed by c_x before entering the holding span, designated value $c_{x,j}$, can be utilized as the current value of c_x in an evaluation of $v_{current\_view}$ until it is known that c_x is no longer in its holding span.

EV3 If a component c_x is non-monotonic, then only its value $c_{x,current\_view}$ can be utilized in the evaluation for $v_{current\_view}$.

EV4 If sufficient information has been received to evaluate an expression of e for $v_{current\_view}$, the value of the expression will be utilized for the evaluation in $v_{current\_view}$ according to the monotonic characteristics of its operands, that is, according to (EV1) if monotonic, according to (EV2) if dependent monotonic and according to (EV3) if non-monotonic.

A recognition for view $v_{current\_view}$ can be characterized as Immediate if the evaluation monitor for the event is located at the same processor as the critical component of the occurrence and, at the time the value of the critical component for view $v_{current\_view}$ is received, all information required for the evaluation of e for $v_{current\_view}$ is present at the evaluation monitor.

7. EXAMPLE INTEGRATION OF ON-THE-FLY EVALUATION TECHNIQUES WITH A VIEW PROTOCOL

The above techniques can be applied to any protocol which is utilized to define consistent views of a computation in order to perform event recognitions. However, for the purposes of discussion, we will assume that *simultaneous regions* are utilized to define such views. Briefly, the technique introduces messages which are sent between an event's monitors in order to define numbered regions. When a change occurs to a monitored component in a region, r, which may result in an occurrence of an event, the component's event monitor is informed of the change. The current values of the components associated with the event monitor are associated with the region number r and the state information is sent to the appropriate evaluation monitor tagged with region number r. Marker messages tagged r are then sent to the other event monitors for the event to signify that a change occurred to a component in region r. Finally, the current region number of the event monitor at which the component whose value changed is located is incremented to $r+1$.

When an event monitor whose components are executing in a region r' receives a region message marked r such that $r' <= r$, the event monitor carries out same steps as are executed if one of its components had experienced a change. If, however, $r' > r$, the message is ignored, since the relationship indicates that the receiving event monitor had already processed its component information for region r and progressed into a subsequent region. Information contained in like-numbered regions is considered simultaneous. Since simultaneous region numbers are maintained by the event monitors, all monitored components at a given processor execute in the same region at any given time. A complete description of the techniques can be found in [14,17].

In order to perform on-the-fly Immediate recognitions, information pertaining to monotonicity must be integrated into the scheme which is used to construct the state views. In particular, the evaluation monitors must be provided with information so that they can deduce if any of the components are monotonic and have reached their holding values, or are dependent monotonic and within a holding span at the time of the evaluation. An evaluation monitor can determine if any components which are local to it are exhibiting monotonic characteristics at the time of the evaluation. However, some means must be provided to communicate this information to evaluation monitors for those components which are not local to the evaluation monitor.

First, let us consider the handling of monotonic components. An evaluation monitor can determine that a component is monotonic and has reached its holding value if either (a) the component is inherently monotonic and the component is known to the evaluation monitor or (b) the component is operator monotonic and the evaluation monitor performs the operation. However, due to the assignment of evaluation responsibilities or the hierarchical definition of an event, an evaluation monitor may not have direct access to knowledge regarding the monotonicity of a particular component. In such a case, a *holding token*, **(component_id,region_number)**, will be generated by the monitor which has access to the information indicating that such a monotonic component has reached its holding value. The first field of the token identifies the component and the second field indicates the region number in which the component reached its holding value. Thus, either through information available locally to an evaluation monitor, or due to the possession of a holding token for a particular component, an evaluation monitor can be provided with sufficient information to determine when a monotonic component has reached a holding value and the region in which the value was assumed. This information is then utilized to determine the applicability of (EV1) of the evaluation algorithm.

To aid in reducing the delay in event recognition, information regarding dependent monotonicity must also be available at the evaluation monitor. Since no other process will be executing at the processor where an evaluation monitor is executing at the time that the monitor is executing, any component located at the same processor where an evaluation monitor is located can be considered dependent monotonic with respect to the activity of the evaluation monitor. However, information must also be available regarding the dependent monotonicity of components located at other processors. In order for an evaluation monitor to be assured that a component is dependent monotonic and within its holding span at the time of evaluation, the dependent element for that component must be located at the same processor as the evaluation monitor. As a means of providing an evaluation monitor with such knowledge, *control tokens* are used. A control token, **<process-id,region_number>**, is associated with each process at which a monitored component resides and is utilized to indicate the locus of control of a process. The first field of the token identifies the process with which it is asso-

ciated and the second field indicates the number of the last region in which the process was active. The control token reflects the control status of a process *P* according to the following rules:

CT1 If process *P* is active, the control token of *P* is located at the processor where *P* is located.

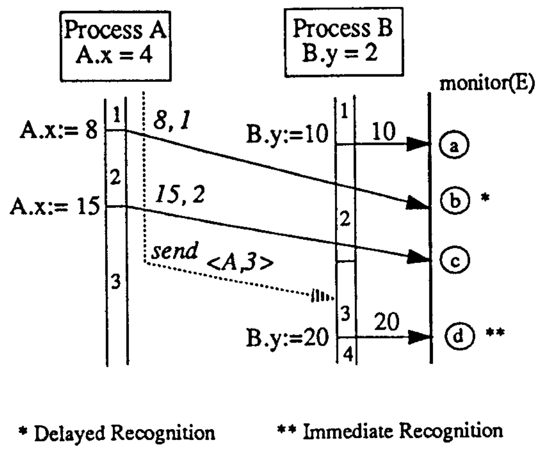
CT2 If process *P* is suspended as a result of an explicit transfer of control (directly or indirectly) to process *P'*, such that *P* can only resume execution as a result of an explicit return of control (directly or indirectly) from process *P'*, then the control token for *P* is held at the processor where *P'* is located.

CT3 If process *P* is suspended as a result of an operation which does not meet the specifications of *CT2*, then the control token of *P* is held at the processor where *P* is located.

Control tokens located at any processor are assumed to be visible to any evaluation monitor located at that processor. The location of the control token for a process *P* at a processor indicates that the dependent element for event components related to *P* is located at that processor. Thus, control tokens provide an evaluation monitor with sufficient information to determine both *explicit* and *implicit* dependent monotonicity. By examining the control tokens located at its processor, an evaluation monitor can determine if any of its components are dependent monotonic and within a holding span at the time of the evaluation.

7.1. EXAMPLES OF ON-THE-FLY RECOGNITIONS

As an example of the application of the on-the-fly evaluation techniques as integrated with view construction via simultaneous regions, consider the activity shown in Figure 3, which depicts the recognition of the event E = *(A.x < B.y)*, that is an event which tests the relationship between the values of an element *x* at process A and and element *y* at process B. It is assumed that the processes reside at different processors. It is also assumed that a rendezvous communication protocol is utilized by the processes A and B. Since it can not be determined, based on the evaluation of the event, which component's change would result in an occurrence of the event, the evaluation will be carried out at the processor where B is located. The evaluation responsibilities for the "less than" test will be incorporated with the event monitor which maintains the state and simultaneous region protocol for B.y. For clarity, only messages which carry information for evaluation are shown. While messages related to the establishment of the simultaneous regions have been omitted, horizontal markings at each process indicate where the region boundaries were formed.



	A.x	B.y	tokens
initial	4_0	2_0	
a		10_1	
b	8_1		
c	15_2	10_2	
d		20_3	<A,2>

Figure 3. Activity while monitoring E = (A.x < B.y).

At the point at which the activity is shown, the value of A.x was 4 in region 0 and the value of B.y was 2. It is assumed that the evaluation monitor has already received this information. First, A.x is assigned the value 8. A.x's new value is associated with region number 1 and the value is sent to the evaluation monitor tagged with the region number. Before A's message reaches the evaluation monitor, B.y is assigned the value 10. This value is transmitted directly to the evaluation monitor. (Region numbering is handled at the monitor, since it also serves as B.y's event monitor.) At the point that B.y's value of 10 for region 1 is received, an evaluation can not be performed since A.x's value of 8 for region 1 has not yet been received. When this information is subsequently received, the evaluation (8 < 10) is performed and an occurrence of the event is recognized. Although the conditions of the event were actually fulfilled at the point that B.y was assigned the value 10, the absence of A.x's state information for region 1 caused a delay between the event occurrence and its recognition.

Another change then occurs to A.x when it is assigned the value 15. This value is captured in region 2 and is again transmitted for evaluation. When this

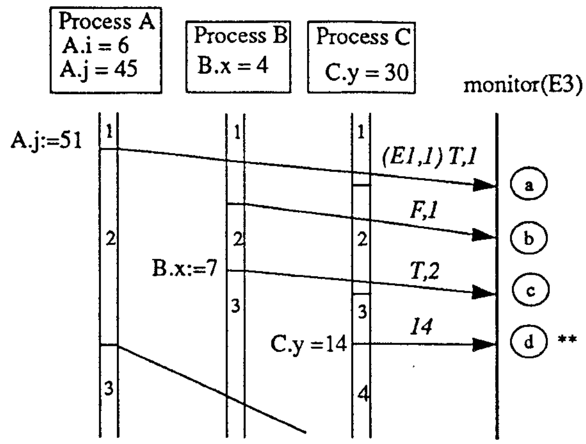
value is received, B.y's value for region 1 is also captured for region 2. The resulting evaluation (15 < 10) yields a value of FALSE, and thus no recognition for region 2 is made by the monitor. Process A then performs a synchronous send to process B. Since A will suspend until B performs a receive, A's control token, marked with the region in which the process suspended, is also transmitted via a monitoring message. (The visibility of this token to an evaluation monitor implies that the state of A, and therefore the value of A.x, would remain at the value which it possessed in the region preceding the transfer of control.) B.y is then assigned the value 20 which is transmitted to the monitor and B.y's new value is associated with region 3. At the point that B.y's value is received by the evaluation monitor, the control token for A is visible to it, guaranteeing that the value of A.x is dependent monotonic and within a holding span. Thus, the value of A.x for region 2 is utilized for the evaluation in region 3. The evaluation (15 < 20) is performed and an occurrence of the event is recognized. Since B.y was the critical component for the event occurrence in region 3, and the value of A.x available to the evaluation monitor located at B's processor could be utilized, an Immediate recognition of the event was achieved.

In the previous example, information indicating that a remote component was dependent monotonic and within a holding span enabled a reduction in recognition time. Consider now, the recognition of the event E3 = *((E1 AND (E2 # (C.y < 25))*. Where event E1 = *(A.i > 10) | (A.j > 50))* and event E2 = *(B.x = 7)*. The responsibility for evaluating the event E3 is assigned to a monitor located at the same processor as C. The event E1 is completely local to A and thus is evaluated and recognized at A. Similarly, the evaluation of E2 is carried out at the processor where B is located. Only the results of these evaluations are transmitted to the evaluation monitor at C. Neither the actual components nor the characteristics of the components of E1 and E2 are known to the monitor at C. The initial values of the individual components are shown in Figure 4. As in the previous example, it is assumed that the values of the components for region 0 are known to the evaluation monitor at this point in the computation and have already been processed.

First, A.j assumes the value 51. This change results in an occurrence of E1, thus causing the value TRUE to be sent to the evaluation monitor for E3 along with the region number, 1, in which it occurred. In addition, E1 is formed using the temporal operator |, causing the components of E1, as well as the event E1, to be operator monotonic. That is, since the event tests an occurrence of its components, the result of TRUE from the evaluation will remain stable. Thus, in addition to

E1 = (A.i > 10) | (A.j > 50) **E2 = (B.x = 7)**

E3 = E1 AND (E2 # (C.y < 25))



	E1	E2	C.y	tokens
initial	F_0	F_0	F_0	
a	T_1		F_1	(E1,1)
b		F_1		
c		T_2	F_2	
d			T_3	

Figure 4. System Activity When Monitoring E3.

the value for region 1, a holding token, **(E1,1)** is also transmitted to the evaluation monitor of E3. The reception of the state information for region 1 causes region 1 to be established for C.y. The change to E1 also causes the value of E2 to be captured for region 1 (although the region protocol messages have been omitted) and its value is sent to the evaluation monitor of E3. Based on the component values for region 1, E3 did not occur and thus there is no recognition.

B.x is then assigned the value 7. This change causes an occurrence and recognition of E2, thus the value of TRUE is transmitted to the the evaluation monitor of E3 for region 2. Since, from the perspective of the monitor at B, event E2 is non-monotonic, no holding token is transmitted. The reception of the message also causes the value of C.y to be associated with region 2. In addition, since a holding token for E1 for region 1 is available, its state value for region 1 can be utilized in an evaluation, eliminating the need to wait for E1's

value for region 2. Thus, an evaluation of E3 is made upon the reception of the message for E2 and found to be FALSE. Finally, an assignment of the value 14 to C.y is made and captured in region 3. This change causes the expression *(C.y < 25)* to be fulfilled. In addition, since E2 is associated with the operator # and the evaluation of that operator is carried out at the evaluation monitor for E3, the evaluation monitor can determine that E2 is operator monotonic with respect to event E3. Thus, E2's value of TRUE for region 2 can be utilized immediately in an evaluation, as can E1's value of TRUE for region 1. The evaluation yields a result of TRUE and thus a recognition of E3 is made for region 3. In addition, since the change to C.y was the critical component for this occurrence and all state information needed to make the recognition was present at the evaluation monitor located at C.y when the change occurred, an Immediate recognition was possible.

8. CONCLUSION

Given the need to accumulate data for the analysis of a distributed computation's activity there can be a considerable time lapse between the occurrence of monitored activity and the analysis of information pertaining to its occurrence. This paper described a two-pronged approach to reducing this delay through the use of techniques which attempt to achieve Immediate event recognitions, that is, which attempt to eliminate any delay, due to inter-processor communication, between an event's occurrence and its recognition by the monitoring system. First, a technique was presented to determine if an Immediate recognition of an event's occurrence could be guaranteed via the placement of evaluation monitors based on the characteristics of an event's operators. Second, techniques were presented which incorporated the stability of an event's components into a strategy for the re-use of component state information. These techniques enable data local to an evaluation monitor to be used in successive evaluations of an event, thus contributing to the reduction of the time required to make a recognition and increasing the possibility that a particular event occurrence can be recognized Immediately.

REFERENCES

[1] P.C. Bates and J.C. Wileden, "High-Level Debugging of Distributed Systems: The Behavioral Abstraction Approach," Journal of Systems and *Software*, vol. 3, no. 4, pp. 255-264, 1983.

[2] P.C. Bates, "Debugging Heterogeneous Distributed Systems Using Event-Based Models of Behavior," *Proceedings of Workshop on Parallel and Distributed Debugging*, pp. 11-22, 1988.

[3] A. Birrel and B. Nelson,"Implementing Remote Procedure Calls", *TOCS*, vol.2, no.1, pp. 39-59, 1984.

[4] K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," ACM Transactions on Computer *Systems*, vol. 3, no. 1. pp. 63-75, 1985.

[5] R. Curtis and L. Whittie, "Bugnet: A Debugging System for Parallel Programming Environments," *Proceedings of the 3rd International Conference on Distributed Computing Systems*, pp. 67-78, 1982.

[6] C. Fidge, "Partial Orders for Parallel Debugging", *Proceedings of Workshop on Parallel and Distributed Debugging*, pp. 183-194, 1988.

[7] D. Haban and W. Weigal, "Global Events and Global Breakpoints in Distributed Systems," Proceedings of 21st Hawaii International Conference of System Sciences, pp. 166-175, 1988.

[8] P.K. Harter, D. Heimbigner, and R. King, "IDD: An Interactive Distributed Debugger",*Proceedings of the 5th International Conference on Distributed Computing Systems*, 1985.

[9] L. Lamport, "Time, Clocks and Ordering of Events in Distributed Systems", *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, July 1978.

[10] R.J. LeBlanc and A.D. Robbins, "Event-Driven Monitoring of Distributed Programs,"*Proceedings of the 5th International Conference on Distributed Computing Systems*, pp. 515-522, 1985.

[11] C.C. Lin and R. LeBlanc, "Event-based Debugging of Object/Action Programs", *Proceedings of Workshop on Parallel and Distributed Debugging*, pp. 23-33, 1988.

[12] B.P. Miller and J.D. Choi, "Breakpoints and Halting in Distributed Systems", *Proceedings of the 8th International Conference on Distributed Computing Systems,*, pp. 316-323, 1988.

[13] E.T. Smith, "Debugging Tools for Message Based, Communicating Processes," *Proceedings of the 4th International Conference on Distributed Computing Systems,*, pp. 303-310, 1984.

[14] M. Spezialetti and J.P. Kearns, "Efficient Global Snapshots," *Proceedings of the 6th International Conference on Distributed Computing Systems*, pp. 382-388, 1986.

[15] M. Spezialetti and J.P. Kearns, " A General Approach to Recognizing Event Occurrences in Distributed Computations", *Proceedings of the 8th*

International Conference on Distributed Computing Systems, pp. 300-307, 1988.

[16] M. Spezialetti and J.P. Kearns, "Simultaneous Regions: An Approach to the Consistent Monitoring of Distributed Computations for Event Occurrences," Proceedings of the 9th International *Conference on Distributed Computing Systems*, pp. 61-69, 1989.

[17] M. Spezialetti, "A Generalized Approach to Monitoring Distributed Computations for Event Occurrences", Ph.D. Dissertation, University of Pittsburgh, 1989.